INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Second Order Function Approximation with a Single Small Multiplication

Jérémie Detrey, Florent de Dinechin

N° 5140

Mars 2004

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--5140--FR+ENG

Second Order Function Approximation with a Single Small Multiplication

Jérémie Detrey, Florent de Dinechin*

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 5140 — Mars 2004 — 13 pages

Abstract: This paper presents a new scheme for the hardware evaluation of elementary functions, based on a piecewise second order minimax approximation. The novelty is that this evaluation requires only one small rectangular multiplication. Therefore the resulting architecture combines a small table size, thanks to second-order evaluation, with a short critical path: Consisting of one table lookup, the rectangular multiplication, and one addition, the critical path is shorter than that of a plain first-order evaluation. Synthesis results for several functions show that this method outperforms all the previously published methods in both area and speed for precisions ranging from 12 to 24 bits.

Key-words: computer arithmetic, harware elementary functions evaluation

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

* ENS-Lyon, LIP (CNRS–ENSL–INRIA–UCBL)

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Approximation de fonction au second ordre avec une seule petite multiplication

Résumé : Cet article présente une nouvelle technique pour l'évaluation en matériel de fonctions élémentaires à partir d'une approximation minimax au second ordre. Son originalité est de ne faire intervenir qu'une petite multiplication rectangulaire. L'architecture résultante combine ainsi la petite taille d'une approximation au second ordre et la vitesse d'une approximation au premier ordre, puisque le chemin critique se compose d'une lecture de table, de la petite multiplication et d'une addition. La synthèse sur FPGA d'opérateurs pour différentes fonctions et différentes précisions montre que cette méthode est plus performante, tant en surface qu'en délai, que toutes les méthodes concurrentes précédemment publiées pour des précisions allant de 12 à 24 bits.

Mots-clés : arithmétique des ordinateurs, fonctions élémentaires en matériel

Second Order Function Approximation with a Single Small Multiplication

Jérémie Detrey , Florent de Dinechin

March 22, 2004

1 Introduction

The evaluation in hardware of elementary functions such as sine/cosine, exp, log, or more complex functions has been an active research subject over the last decade. Applications include digital signal processing, but also neural networks [15], logarithm number system [5], and the initialization of Newton-Raphson iterations for hardware division [11] among many others.

The simplest hardware evaluator is a lookup table storing precomputed values. Its size grows exponentially with the size of the input word, which confines this solution to input precisions smaller than 10 bits. The table size can be reduced by using a piecewise linear approximation of the function. The hardware now requires a multiplier [10], but the bipartite trick and its variations [2, 14, 12, 3] allow to replace the multiplier with an adder, which improves both area and speed. These methods allow for practical input precisions up to 20 bits. For more precision, approximations by higher order polynomials are needed [7], using either more multipliers, or iterations over a single multiplier, with an increased delay. Variations on these higher order methods include partial product arrays [6], parallel powering units [13, 9], and difference formulas [8, 1].

This article presents a second order method which involves only one small rectangular multiplication. This method is well suited to input precisions from 10 to 24 bits. It is simpler and more flexible than previous similar work [4], allowing complete automation of the synthesis of operators for arbitrary functions and arbitrary input and output precision. This scheme also outperforms the other previously published methods listed above in both area and speed.

*ENS-Lyon, LIP (CNRS–ENSL–INRIA–UCBL)

Notations

Throughout this paper, we discuss the implementation of a function whose inputs and outputs are in fixed-point format. We note w_I and w_O the required input and output size (in bits). Without loss of generality, we will focus in this paper on functions with both domain and range equal to $[0;1[$. Thus any input word X is written $X = .x_1x_2\cdots x_{w_I}$ and denotes the value $\sum_{i=1}^{w_I} 2^{-i}x_i$. Similarly an output word is written $Y = .y_1y_2\cdots y_{w_O}$.

2 The SMSO approximation scheme

2.1 General idea

The main idea behind the Single Multiplication Second Order method (SMSO) is to consider a piecewise degree 2 polynomial approximation of the function f. The input word X is thus split into two sub-words A and B of respective sizes α and β, with $\alpha+\beta = w_I$ (see Figure 1) :

$$X = A + 2^{-\alpha}B = .a_1a_2\cdots a_\alpha b_1b_2\cdots b_\beta.$$

The input domain is split in 2^α intervals selected by A. On each of these intervals, f is approximated by a second order polynomial:

$$\begin{aligned} f(X) &= f(A+2^{-\alpha}B) \\ &\approx K_0(A) \;+\; K_1(A)\times 2^{-\alpha}B \;+\; K_2(A)\times 2^{-2\alpha}(B-\tfrac{1-2^{-\beta}}{2})^2. \end{aligned}$$

Remark: we need the parabolic component to be centered in the interval so that we can exploit symmetry later on.

We can then split B into two sub-words B_0 and B_1 of respective sizes β_0 and β_1, with $\beta_0+\beta_1=\beta$ (see Figure 1). In other words $B = B_0 + 2^{-\beta_0}B_1$. This gives:

$$\begin{aligned} f(X) \;\approx\; & K_0(A) \\ & + K_1(A)\times 2^{-\alpha}B_0 \;+\; K_1(A)\times 2^{-\alpha-\beta_0}B_1 \\ & + K_2(A)\times 2^{-2\alpha}(B-\tfrac{1-2^{-\beta}}{2})^2. \end{aligned} \tag{1}$$

We decide to tabulate as follows:

- A *Table of Initial Values*: $\mathrm{TIV}(A) = K_0(A)$;
- A *Table of Slopes*: $\mathrm{TS}(A) = 2^{-\alpha}K_1(A)$;
- Two *Tables of Offsets*: $\mathrm{TO}_1(A,B_1) = 2^{-\alpha-\beta_0}K_1(A)\times B_1$ and $\mathrm{TO}_2(A,B) = 2^{-2\alpha}K_2(A)\times (B-\frac{1-2^{-\beta}}{2})^2$.

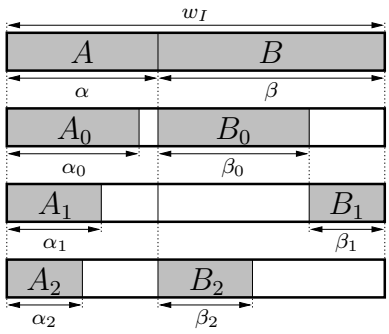


Figure 1: Decomposition of the input word X.

We then have:

$$f(X) \approx \text{TIV}(A) + \text{TS}(A) \times B_0 + \text{TO}_1(A, B_1) + \text{TO}_2(A, B)$$

where there is only one multiplication, the rest being table lookups and additions.

In this scheme so far, the approximation error is only due to the initial polynomial approximation. Remark, however, that the relative accuracies of the various terms are different, due to the powers of two in Eq. 1. We may therefore degrade the accuracy of the most accurate terms (the least significant ones), to align it on the least accurate terms. This is achieved by reducing the number of bits addressing the various tables, which will reduce their size. Section 3 will quantify this relation between the approximation error and the various parameters (the function, w_I, w_O, α, β, the α_i, the β_i, and the internal precision used), which determine the table and multiplier sizes.

- The TS is addressed by $A_0 = .a_1 a_2 \cdots a_{\alpha_0}$ the $\alpha_0 \leq \alpha$ most significant bits of A.
- The TO_1 is addressed by $A_1 = .a_1 a_2 \cdots a_{\alpha_1}$ the $\alpha_1 \leq \alpha$ most significant bits of A and B_1.
- The TO_2 is addressed by $A_2 = .a_1 a_2 \cdots a_{\alpha_2}$ the $\alpha_2 \leq \alpha$ most significant bits of A, and $B_2 = .b_1 b_2 \cdots b_{\beta_2}$ the $\beta_2 \leq \beta$ most significant bits of B.

Finally, we get the SMSO approximation formula below, which can be implemented as the architecture depicted Fig. 3:

$$f(X) \approx \text{TIV}(A) + \text{TS}(A_0) \times B_0 + \text{TO}_1(A_1, B_1) + \text{TO}_2(A_2, B_2)$$

2.2 Exploiting symmetry

As remarked by Schulte and Stine in [14] in the case of the multipartite method, the tables present some symmetry. We have $\text{TO}_1(A_1, B_1) = 2^{-\alpha-\beta_0} K_1(A_1) \times B_1$, which can be rewritten:

$$\begin{aligned} \mathrm{TO}_1(A_1, B_1) &= 2^{-\alpha-\beta_0} K_1(A_1) \times B_1 \\ &= 2^{-\alpha-\beta_0} \left(K_1(A_1) \times (B_1 - \tfrac{1-2^{-\beta_1}}{2}) + K_1(A_1) \times \tfrac{1-2^{-\beta_1}}{2} \right) \end{aligned}$$

where the term $2^{-\alpha-\beta_0} K_1(A_1) \times \frac{1-2^{-\beta_1}}{2}$ can be added to the value of the TIV. This allows us to use the segment symmetry as depicted in Fig. 2, saving a bit in addressing the TO_1 at the expense of a few XOR gates needed to reconstruct the other half of the segment.

The values of TO_2 also present symmetry, which allows to divide its size by two as well. In this case the output of the table should not be XORed, as TO_2 holds an even function (see Fig. 3).

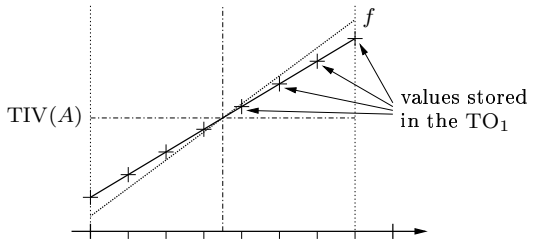


Figure 2: Example of segment symmetry.

2.3 Architecture

An example of SMSO operator architecture is given Fig. 3. All the table lookups are performed in parallel. One should also notice that two of the three additions of the adder tree can be performed in parallel to the multiplication. Therefore the critical path is the TS table lookup, the multiplication and the last addition.

An important advantage of this scheme is that the multiplier is kept small and rectangular due to the splitting of B. This will lead to efficient implementation on current FPGA hardware with fast carry circuitry (and even more efficient if block multipliers are available).

The remainder of this article shows how to choose the numerous parameters introduced here to ensure a given accuracy bound.

3 Optimisation of SMSO operators

In the following, we want a SMSO architecture to (classically) provide *faithful* accuracy: The result returned must be one of the two numbers surrounding the exact mathematical result, or in other terms, the total error of the scheme should always be strictly smaller than 2^{-w_O}. However all the following is easily adapted to other error bounds. The bound on the global error ϵ made by the SMSO operator is the sum of several terms:

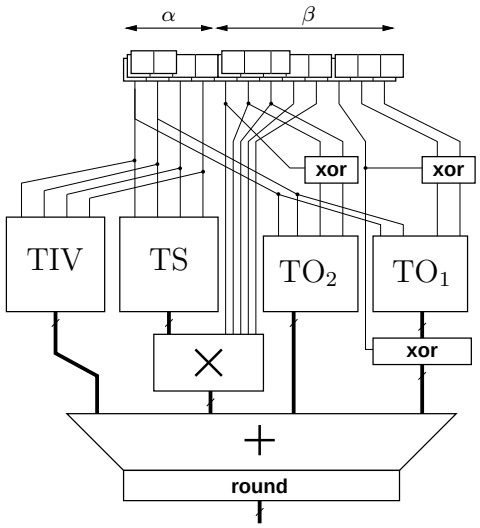


Figure 3: Architecture of the SMSO operator for $\alpha = 4$, $\beta = 8$, $\alpha_0 = \alpha = 4$, $\alpha_1 = \alpha_2 = 2$, $\beta_0 = 5$, $\beta_1 = 3$ and $\beta_2 = 3$

$$\epsilon = \epsilon_{\text{poly}} + \epsilon_{\text{tab}} + \epsilon_{\text{rt}} + \epsilon_{\text{rm}} + \epsilon_{\text{rf}},$$

where:

- ϵ_{poly} is the error due to the polynomial approximation, studied in 3.1;
- ϵ_{tab} is the approximation error due to removing bits from the table inputs as shown previously; It is studied in 3.2;
- ϵ_{rt} and ϵ_{rf} are rounding errors, when filling the tables, the product and the final sum; They are studied in 3.3;

In the following, we show how these terms can be computed, depending on the design parameters. An heuristic for optimising a SMSO operator then consists in enumerating the parameter space, computing the error for each value of the parameters, keeping only those which ensure faithul accuracy, and selecting among them the optimal either in terms of speed or of area.

3.1 Polynomial coefficients - ϵ_{poly}

The coefficients $K_0(A)$, $K_1(A)$ and $K_2(A)$ are computed on each of the 2^α intervals as a minimax approximation based on the Remez algorithm[11]. This method provides us with the 3 coefficients along with the value of ϵ_{poly}. To cut the exploration of the parameter

space, we may remark that this error is obviously bounded by the second order Taylor approximation error:
$\epsilon_{\text{poly}} \leq \frac{1}{6} 2^{-3\alpha-3} \max_{X \in [0;1[} |f'''(X)|$

3.2 Reducing table input sizes - ϵ_{tab}

Removing $\alpha - \alpha_i$ bits from the input of one table means imposing a constant table value over an interval of size $2^{\alpha-\alpha_i}$. As the content of the table is usually monotonous, the value that minimises the error due to this approximation is the mean of the extremal values on this interval, and the error induced is then the half of the distance between these extremal values, suitably scaled according to Eq. 1.

The symmetry reduction described in Section 2.2 to halve the size of the TO_is entails no additional approximation error.

3.3 Rounding considerations - ϵ_{rt} and ϵ_{rf}

Unfortunately, the tables cannot be filled with results rounded to the target precision: Each table would entail a maximum rounding error of 2^{-w_O-1}, exceeding the total error budget of 2^{-w_O}. We therefore fill the TIV and the TO_is with a precision greater than the target precision by g_0 bits (guard bits). Thus rounding errors in filling one table is now $2^{-w_O-g_0-1}$ and can be made as small as desired by increasing g_0. For consistency of the final summation we chose to round the output of the multiplier to g_0 bits as well, by truncating it and adding half a bit to the value in the TIV before rounding . Thus the total error due to these four roundings is bounded by $4 \times 2^{-w_O-g_0-1} = 2^{-w_O-g_0+1}$.

The output of the TS table is not concerned by the previous discussion, and we may control its rounding error by another number of guard bits g_1. This entails another rounding error that adds up with the summation errors. Finally we have:

$$\epsilon_{\text{rt}} = 2^{-w_O-g_0+1} + 2^{-w_O-g_1-1} \quad .$$

The final summation is now also performed on g_0 more bits than the target precision. Rounding the final sum to the target precision now entails a rounding error up to $\epsilon_{\text{rf}} = 2^{-w_O-1}$. A classical trick due to Das Sarma and Matula [2] allows to improve it to $\epsilon_{\text{rf}} = 2^{-w_O-1}(1 - 2^{-g_0})$.

Note that this discussion has added another two parameters g_0 and g_1 to the SMSO architecture. Currently, the values of g_0 and g_1 are computed by trial-and-error, increasing them while the accuracy bound is not reached.

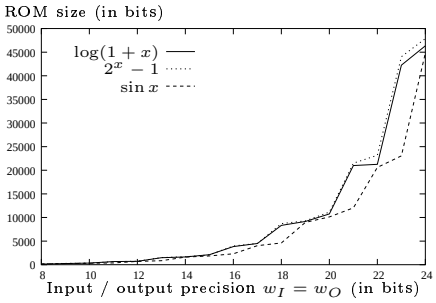
There is an implicit implementation choice in the previous error analysis, which is that we use an exact, full-precision multiplier. Another option would be to truncate the multiplier hardware directly. Our choice is obvious when targetting FPGAs with small multipliers, like the Virtex-II. It also makes sense in the other cases, as it allows to cleanly express the error as a function of the parameters. Besides, the expected gain in using a truncated multiplier is less than half the size of the multiplier, which is itself small compared to the tables as Section 4 will show. Therefore this choice seems justified a-posteriori.

4 Results

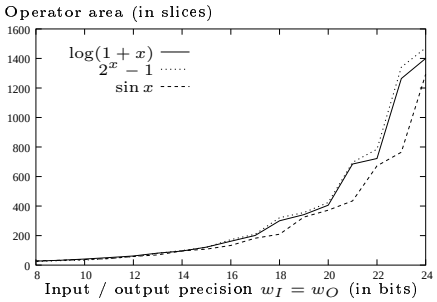
4.1 ROM size, area and delay estimations

In this section we give estimations of area and critical path delay for varying precisions (with $w_I = w_O$) for the following three functions:

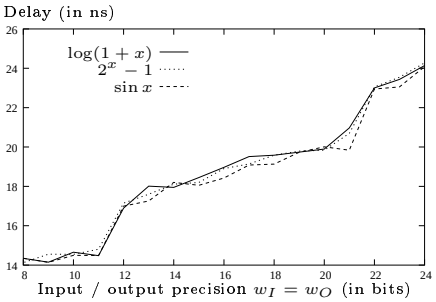
- The natural logarithm: $\log(1+x) : [0;1[\rightarrow [0;1[$;
- The power of 2: $2^x - 1 : [0;1[\rightarrow [0;1[$;
- The sine: $\sin(\frac{\pi}{4}x) : [0;1[\rightarrow [0;1[$.



(a) Size of the tables



(b) Operator area



(c) Critical path delay

Figure 4: Area and delay of some SMSO operators (without using block multipliers).

These estimations were obtained using Xilinx ISE v5.2 for a Virtex-II XC2V1000-4 FPGA. We performed synthesis with and without using the small multipliers embedded in those FPGAs, to compare our results with those of other published works. Only results

Function		$\log(1+x)$			$\sin x$		
Precision ($w_I = w_O$)		16 bits	20 bits	24 bits	16 bits	20 bits	24 bits
Multiplier bit size		6×9	8×14	10×16	6×12	8×14	10×15
not using block multipliers	area (slices)	162	406	1400	133	372	1293
	delay (ns)	19	20	24	18	20	24
using block multipliers	area (slices)	135	349	1318	98	315	1218
	delay (ns)	16	18	20	16	18	20

Table 1: Impact of using the Virtex-II 18×18 block multipliers.

for combinatorial operators are detailed, as the estimations for pipelined circuits present only slight differences.

Fig. 4(a) shows that the combined size of the four tables (TIV, TS and TO_is) grows exponentially with the precision, as expected for a table-based method. Fig. 4(b) closely resembles Fig. 4(a), which indicates that the adders and multiplier contribute only by a small amount to the overall area of the operators. This fact is also underlined by Table 1, which studies the impact on area and delay of using block multipliers: the difference in area corresponds roughly to the area of the multiplier implemented in slices.

Fig. 4(c) shows that the delay of the SMSO operators grows linearly with the precision, as it is dominated by the table lookup delay which is logarithmic in the size of the tables. For precisions up to 24 bits, the SMSO operators can run at frequencies higher than 33 MHz. Pipelined designs in 3 to 4 stages have been successfully tested at 100 MHz. Table 1 shows that using the small multipliers provided by Virtex-II FPGAs speeds up the whole circuit by 10 to 20%.

As a conclusion, implementing these operators on FPGAs provided with small multipliers will bring improvements in both area and speed, but performance is still very close without embedded multipliers, so the method is also well-suited to multiplier-less FPGA families.

4.2 Comparison with previous works

We first compare our SMSO scheme to the state of the art in multipartite method by de Dinechin and Tisserand [3]. Table 2 shows that, thanks to its 2nd-order approximation, a SMSO operator is always much smaller than its (first-order) multipartite counterpart. On Virtex FPGAs, this gain in size also allows our method to outperform the multipartite method in terms of delay, despite the multiplier in the critical path where the multipartite scheme have only additions.

We also compare our method to the lookup-multiply units developed by Mencer et al. in [10]. The results they publish are obtained on XC4000 FPGAs, which prevents comparing delays. As XC4000 CLBs can be compared to Virtex-II slices, Table 3 shows that the SMSO operators are much smaller than lookup-multiply units, which actually seem less efficient than multipartite ones.

Function		$\sin x$					$2^x - 1$
Precision ($w_I = w_O$)		8 bits	12 bits	16 bits	20 bits	24 bits	16 bits
Multipartite	table size (bits)	—	—	7808	—	189440	8704
	area (slices)	19	76	258	1209	4954	283
	delay (ns)	17	18	24	34	43	23
SMSO	table size (bits)	134	704	2304	10112	44800	3968
	area (slices)	28	59	133	372	1293	173
	delay (ns)	14	17	18	20	24	19

Table 2: Compared table size, area and delay of the multiparitite table method [3] and of the SMSO for the $\sin x$ and $2^x - 1$ functions.

Precision ($w_I = w_O$)	8 bits	12 bits	16 bits	20 bits	24 bits
Lookup-multiply area (XC4000 CLBs)	80	180	560	2000	8900
SMSO area (Virtex-II slices)	27	62	162	406	1400

Table 3: Area compared to the lookup-multiply method [10] for the $\log(1 + x)$ function.

Finally, we want to compare the SMSO scheme with the faithful powering computation developed by Piñero et al. in [13], once again implemented on XC4000 FPGAs. Their method uses a squarer unit and a multiplier to compute a second-order approximation, which is probably more generally applicable than what they publish: Their architecture is hand-crafted for powering functions with a precision of 23 bits. Their area estimation (1130 slices, but it is unclear for which function) is roughly the same as those of our method (about 1000 slices, depending on the function), but their critical path is larger, as their operator performs all the additions after the multiplications. Besides the strong point of our method here is its flexibility.

5 Conclusion

We have presented a new scheme for elementary function approximation, based on a piecewise degree 2 minimax approximation involving only one small rectangular multiplication. The method is simple and leads to architectures well suited to modern FPGAs, is suitable for any function and any precision, and performs better in terms of area and speed than all previously published methods for hardware function evaluation in the precision range 12-24 bits. For smaller precisions, a simple table or the multipartite method may be more efficient.

The current weakness of our work is that the choice of the many parameters does not involve an exhaustive exploration of the parameter space depicted in Section 3: our current heuristic finds an architecture that ensures correct rounding, but does not guarantee it is the most efficient one. Therefore the full flexibility of the method is not exploited, as illustrated

by the similarities of the graphs for the three functions on Fig. 4. Our efforts now concentrate on the design of an efficient heuristic for exploring this parameter space. Of course, this can only improve our results. If block multipliers are available, their metrics should be taken into account in our heuristic.

This work will also lead to improvements in our LNS operator library [5].

References

[1] J. Cao, B.W.Y. Wei, and J. Cheng. High-performance architectures for elementary function generation. In Neil Burgess and Luigi Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, June 2001.

[2] D. Das Sarma and D.W. Matula. Faithful bipartite ROM reciprocal tables. In S. Knowles and W.H. McAllister, editors, *12th IEEE Symposium on Computer Arithmetic*, pages 17–28, Bath, UK, 1995. IEEE Computer Society Press.

[3] F. de Dinechin and A. Tisserand. Some improvements on multipartite table methods. In Neil Burgess and Luigi Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic*, pages 128–135, Vail, Colorado, June 2001. Updated version of LIP research report 2000-38.

[4] D. Defour, F. de Dinechin, and J.M. Muller. A new scheme for table-based evaluation of functions. In *36th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, November 2002.

[5] J. Detrey and F. de Dinechin. A VHDL library of LNS operators. In *37th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, USA, October 2003.

[6] H. Hassler and N. Takagi. Function evaluation by table look-up and addition. In S. Knowles and W.H. McAllister, editors, *12th IEEE Symposium on Computer Arithmetic*, pages 10–16, Bath, UK, 1995. IEEE Computer Society Press.

[7] D-U Lee, W. Luk, J. Villasenor, and P. Cheung. Hierarchical segmentation schemes for function evaluation. In *IEEE Conference on Field-Programmable Technology*, Tokyo, dec 2003.

[8] D.M. Lewis. Interleaved memory function interpolators with application to an accurate LNS arithmetic unit. *IEEE Transactions on Computers*, 43(8):974–982, August 1994.

[9] A.A. Liddicoat. *High-performance arithmetic for division and the elementary functions.* PhD thesis, Stanford University, 2002.

[10] O. Mencer, N. Boullis, W. Luk, and H. Styles. Parametrized function evaluation on fpgas. In *Field-Programmable Logic and Applications*, Belfast, September 2001.

[11] J.M. Muller. *Elementary Functions, Algorithms and Implementation.* Birkhauser, Boston, 1997.

[12] J.M. Muller. A few results on table-based methods. *Reliable Computing*, 5(3):279–288, 1999.

[13] J. A. Piñeiro, J. D. Bruguera, and J.-M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In Neil Burgess and Luigi Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic*, pages 40–47, Vail, Colorado, June 2001.

[14] J.E. Stine and M.J. Schulte. The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing*, 21(2):167–177, 1999.

[15] S. Vassiliadis, M. Zhang, and J. G. Delgado-Frias. Elementary function generators for neural-network emulators. *IEEE transactions on neural networks*, 11(6):1438–1449, nov 2000.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399